On Automated Graph Layout

AN ALGORITHMIC AND A DECLARATIVE APPROACH DESIGNED AND IMPLEMENTED IN THE FUNCTIONAL SYSTEM CLARITY

JUNE 2001

ROGIER VAN DALEN AND MIKE SPAANS

Table of Contents

Acknowledgements

We would like to thank Dr. L.J.M. Rothkrantz, Delft University of Technology, for providing us with the opportunity to visit Portsmouth.
We would like to thank Prof. T.R. Addis, Portsmouth University, for supervising our research, supporting our activities and providing ideas.

Portsmouth, June 6, 2001
Rogier van Dalen
Mike Spaans
Delft University of Technology

Abstract

The functional development environment *Clarity* uses *network diagrams* to provide an overview of how functions are dependent. Currently, the environment does not feature an option to automatically arrange the functions in this diagram in such a way that the structure is visible. Two approaches are discussed: Simulated Annealing (a declarative method) and Sugiyama (an algorithmic method). The Sugiyama method appears to be the more suitable approach for it takes less computing time and produces easier readable diagrams by introducing a hierarchical structure.

1 Introduction

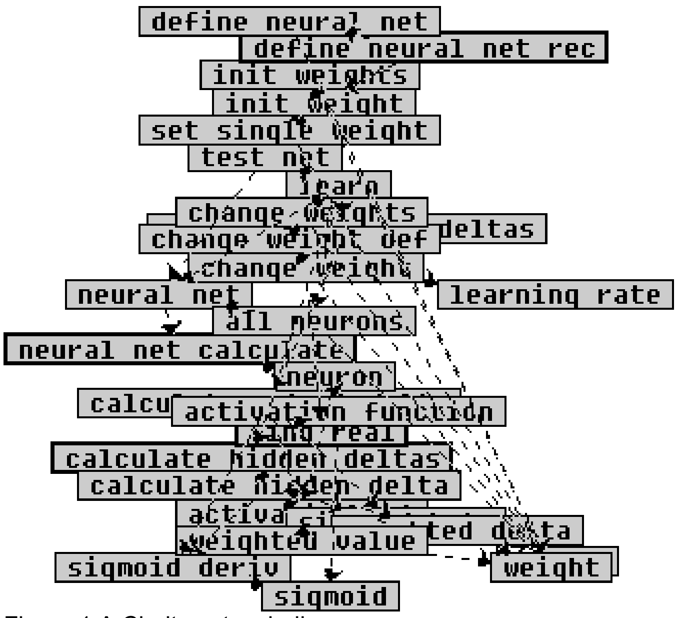

Figure 1 A Clarity network diagram

Figure 1 depicts a *network diagram* in the visual development environment Clarity, as the system currently creates them automatically. The purpose of network diagrams is to provide an overview of how the boxes (representing functions in Clarity) are dependent. While users are able to create satisfactory layouts in a short time by moving the boxes around, for computers this is an exceptionally difficult problem to solve[1]. However, extensive research has been done on similar problems, such as placement of labels on geographical maps [1] and good results have been attained. A number of methods for solving these problems have been devised, two of which will be discussed: Simulated Annealing, which is a declarative method, and the Sugiyama algorithm, which is an algorithmic method.
In the following sections an overview of the development environment Clarity will be presented, the algorithm of Simulated Annealing and Sugiyama and their implementations will be discussed and finally these two methods will be analysed in terms of their complexity and behaviour.

[1] As it turns out, this problem, due to its combinatorial nature, is part of a class of problems which cannot be solved in polynomial time, i.e., NP-hard.

2 Clarity

2.1 Functional programming

Throughout the programming community, a number of programming paradigms are in use, the imperative paradigm being the most popular and widespread. There are, however, other paradigms, such as the *functional paradigm*. Functional programming is a style of programming that emphasises the evaluation of expressions, rather than execution of commands. The expressions in these languages are formed by using functions to combine basic values. Because these languages are based on mathematics and on mathematical ways of expressing functions, functional programming languages are often conceived as being difficult to use.
“Purely functional” programming languages do not allow functions with *side effects*, such as writing data to the screen, getting user input or function redefinition. The functional system Clarity does not have this restriction.

2.2 Clarity

Clarity is a graphical development environment, based on a functional programming language. In Clarity functions are drawn on the screen: different components of a function, such as parameters, constants, built-in functions, etc, are placed on the screen, and their outputs and/or inputs connected by arrows (see Figure 2).
The underlying functional language is called *Faith*. Faith code can be generated automatically from the function pictures. The Faith code can then be run by the interpreter.

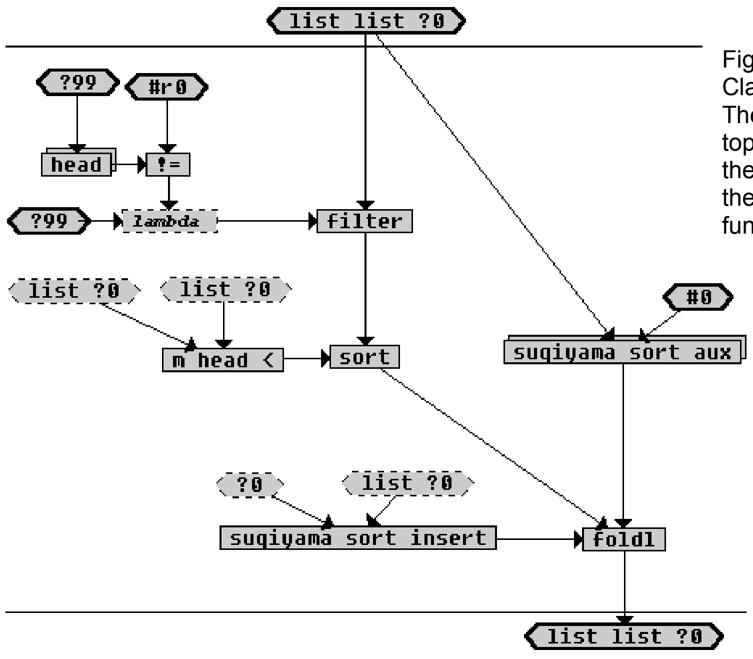

Figure 2 An example of a function in Clarity.
The lozenge-shaped element at the top is a parameter lozenge, the one at the bottom the function return value, the others are constants. Boxes are functions.

3 Graph Layout

In this section the problem of automated layout of network diagrams is formalised by introducing readability constraints and two approaches to solving this problem are suggested.

3.1 Requirements Imposed on Network Diagrams

In order to use computer algorithms to arrange network diagrams readability constraints of Clarity network diagrams have to be formalised. These constraints are:

1. Lines should not cross, as crossing lines make it more difficult to quickly see the connections between boxes.
2. Lines should not cross boxes, as this makes it difficult to see whether the line is starting at the box or crossing the box.
3. Hierarchy should be apparent: higher-level functions should be placed higher in the diagram than lower-level functions. A box that has an arrow pointing to another box has a higher level than the other box, because the arrow signifies that the function in the former box uses the function in the latter. This will result in all arrows pointing down.
4. It is desirable for arrows to be short, to make it easier for the reader to follow them.
5. It is desirable that arrows are aligned either horizontally or vertically, making it easier for the reader to follow them.
6. It is desirable that the distance between any two boxes is relatively short, so that the diagram is easy to overview.
7. It is desirable that the area a network takes up is small, making the diagram easy to read at a glance.

To provide a reference to the importance of these constraints, they are structured in Table 1, in which two dimensions have been defined. Constraints placed on the left in the diagram are more structural in nature than constraints placed on the right: the former have to do with the structure of the graph as a whole, while the latter can be imposed after the general structure is defined and only slight changes have to be made. The distinction between required and merely desired readability constraints is made by the other dimension. Constraints placed at the top of the diagram are of higher priority than constraints placed at the bottom.

	structural		aesthetic
high priority	crossing lines (1)	box line (2)	
	hierarchy (3)		distance (6)
			window size (7)
		arrow length (4)	
low priority			straight lines (5)

Table 1

3.2 Approaches to the Problem

As the problem of automated layout of network diagrams is combinatorial in nature, to solve the problem in reasonable time approximation algorithms have to be devised. In general graph drawing, two approaches are used to automate graph layout: the *declarative* approach and the *algorithmic* approach [3].

The declarative approach is the most direct. In this approach, the layout of the graph is specified by a user-defined set of constraints, and it is generated by solving the system of constraints. While this is a direct and expressive way of formalising constraints into a computer program,

the program itself usually takes a long time to compute a graph layout and is generally inefficient.
In the algorithmic approach, the idea is to devise an algorithm that is supposed to produce a graph that meets a prespecified set of criteria. Though this approach is usually computationally efficient, it is more difficult to meet all criteria, and changing criteria or changing priority usually involves developing a new algorithm. Because the algorithm usually runs polynomial time, it is able to deal with large problems.

4 Simulated Annealing

In the following sections the Simulated Annealing algorithm will be discussed in greater detail. The manner in which the algorithm can be applied to the specified layout problem of Clarity network diagrams and an implementation will also be discussed.

4.1 Theory

Simulated Annealing[2] (SA) is a stochastic optimisation algorithm, which is used to find the minimum of a function. It uses a declarative approach for which the constraints are implemented using a *cost function*: a function whose result increases when the outcome of the random permutation is more desirable and decreases when the outcome is less desirable. SA is implemented as follows.
The *state* of the diagram (the positions of the boxes) is changed randomly at every step. The new state is either accepted or rejected, depending on the energy level (the cost function). Two variables are used: the amount by which the state is changed randomly (*P*), and the probability with which a state with a higher energy level than the previous one is accepted. The latter is called the 'temperature' *T* of the system. The energy level is defined as a function of the state (the cost function). The reason states with higher energy levels are accepted at all, is to make it possible to work around local minima. To find the global minimum, the state is changed randomly to find a lower energy level than the previous. Over time, however, the amount the state is changed should converge to zero, and so should the probability with which a state higher than the current is accepted, as shown in (1) and (2).

$$P_{new} = \lambda \cdot P_{old} \text{ with } 0 \leq \lambda_P < 1 \qquad (1)$$

$$T_{new} = \lambda \cdot T_{old} \text{ with } 0 \leq \lambda_T < 1 \qquad (2)$$

It is clear both the temperature *T* and the amount of change *P* should decrease over time, but the speed this happens at depends on the instance of the problem and can only be found out through experiment.
The function which global minimum is to be found is called the cost function (Γ). This function is arbitrarily defined. In the specified layout problem, the cost is higher if undesirable effects such as overlap occur.

4.1.1 Explicit Behaviour of the SA Algorithm in Terms of the Γ function

In this section an overview will be given of how the cost function (Γ) of the Simulated Annealing algorithm for the specified layout problem has been designed. In order to determine what a good Clarity network diagram looks like, readability constraints of network diagrams have been elicited and formalised in paragraph 3.1. For all readability constraints elicited, a sub-function will be defined. The sub-functions all return values that contribute to the total cost of the system. The sub-functions are summed to form Γ as shown in (3).

$$\Gamma = \sum k_i \Gamma_i \qquad (3)$$

- $\Gamma_{overlap}$
  The value returned by $\Gamma_{overlap}$ is the total area of any box overlapping any other box in the diagram.

[2] The term Simulated Annealing comes from the field of metallurgy where annealing is the process of repeated slow cooling of steel to increase the extent of the crystal structure.

- $\Gamma_{distance}$

  The $\Gamma_{distance}$ sub-function calculates the distance between all the boxes in the diagram. The returned value is the sum of the distances corrected for an optimal distance. The following function defines this correction:

  $$|x-a| \qquad (4)$$

  where a is the optimal length of the arrows.

- $\Gamma_{windowsize}$

  To prevent the diagram from becoming too large, a cost will be calculated for the minimum window size the boxes will fit in. The $\Gamma_{windowsize}$ sub-function is calculated by adding the width and the height for the window size.

- $\Gamma_{crossing\_lines}$

  The $\Gamma_{crossing\_lines}$ sub-function returns a value based on the number of lines crossing. The associated cost is based on a Gaussian function for every crossing:

  $$\frac{1}{\sqrt{2\pi\sigma}} \cdot e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \qquad (\mu = 0, \sigma = 1) \qquad (5)$$

  in order to obtain the highest cost when the lines cross in their respective centres, and decreases when they cross at the edges. This is done so that the SA algorithm can generate states which cross less each time, until no crossings are left.

- Γ_{arrows_down}

  For the diagram to show the underlying hierarchy, the top-level functions should be at the top, while the lower-level functions should be lower. This can be achieved by making the arrows point downward: as top-level functions use lower-level functions, arrows go from higher-level to lower-level functions. This is achieved by increasing the cost whenever the arrow points higher, as shown in Figure 3. The cost is based on the following formula for every arrow:

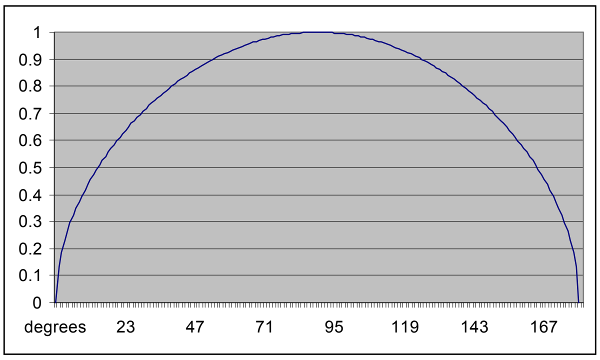

Figure 3 Behaviour of Γ_{arrows_down}

  $$\sqrt{\frac{\Delta y}{eucl\_dist(\Delta x, \Delta y)}} \qquad (6)$$

- $\Gamma_{straight\_lines}$

  It is desirable for the arrows to be either vertically or horizontally, as this improves the diagram. This is done by increasing the cost for arrows with angles other than 0°, 90°, 180° and 270°. The cost is based on the following:

  $$|\sin(2 \cdot \arctan(slope))|$$

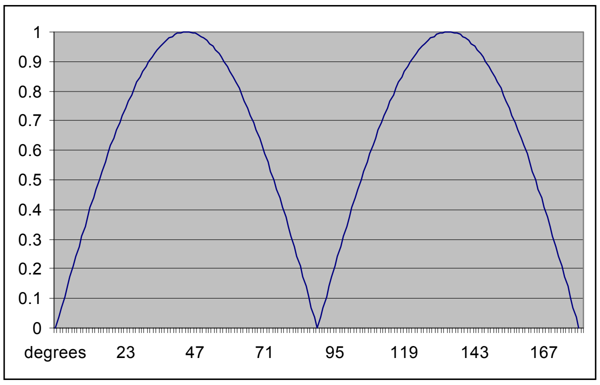

Figure 4 Behaviour of $\Gamma_{straight\_lines}$

- Γ_{box_line}
  Lines should not cross boxes, because the contents of the box will be more difficult to read and it will not be clear whether the line ends or starts at the box or just crosses the box. Any line that passes through the box will increase the cost. When the line crosses the middle of the box, the cost will be higher. This behaviour is implemented using a Gaussian function.

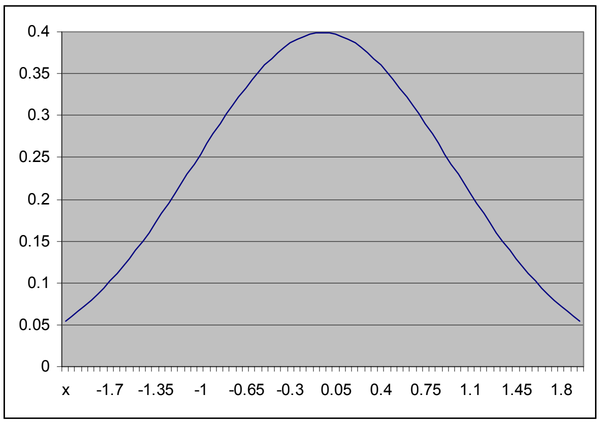

Figure 5 Behaviour of Γ_{box_line}

- $\Gamma_{arrowlength}$
  The $\Gamma_{arrowlength}$ sub-function returns a value based the length of the arrows. Each arrow contributes a cost if it is longer or shorter than the desired optimal length. This is implemented as shown in (7).

$$|x-a|^b \tag{7}$$

where a is the optimal arrow length, and b the parameter which defines cost increase steepness.

4.2 Implementation

In this section, an overview will be given of the implementation of the Simulated Annealing algorithm in the functional development environment Clarity. The implementation consists of two parts: the actual Simulated Annealing algorithm and the cost function Γ.

4.2.1 The Simulated Annealing Algorithm

In order to use information about the position and size of boxes and arrows for the Simulated Annealing, the positions of the boxes are converted to a list of integers. The list contains the horizontal and vertical positions of the boxes. When the cost function is calculated or the diagram is displayed, the boxes are moved to their positions given in the list of integers, and the arrows are positioned accordingly.
A network diagram of the simulated annealing algorithm is shown in Figure 6.

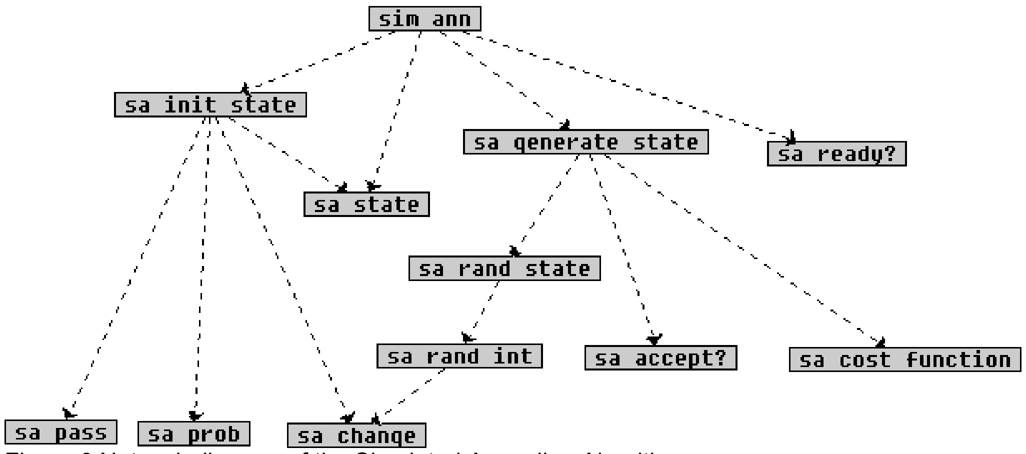

Figure 6 Network diagram of the Simulated Annealing Algorithm

The `sim_ann` function is the main function. First, `sa_init_state` is called to initialise both `sa_prob` (*T*), `sa_change` (*P*) and `sa_state` (the state as a list of integers). `sim_ann` then repeatedly calls `sa_generate_state` until `sa_ready?` returns `True` (when the number of iterations equals an initially given number).
`sa_generate_state` takes a parameter, the current state as a list of integers, and generates a new state based upon the old one. The cost of the new state (calculated by `sa_cost_function`) is compared to the previous state. If the new cost is lower than the previous, the new state is accepted with probability 1. Otherwise, the new state is accepted with probability *T* (`sa_prob`).

4.2.2 Cost Function

The network diagram for the cost function is given in Figure 7.

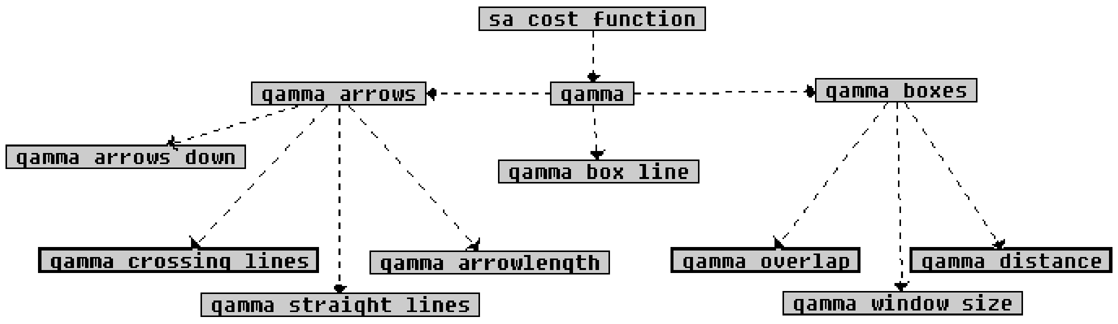

Figure 7 Network diagram for the cost function

To accommodate for the difference between the representation of the Simulated Annealing algorithm and a Clarity diagram, the list of integers that represent postitions is converted by `sa_cost_function`, which passes a list of `objects` (boxes and arrows) to `gamma`.
Γ sub-functions are normalised and multiplied by a weight. Normalisation is needed to make sure the relationships between sub-functions are maintained when the diagram contains more or less boxes. The result of every sub-function is divided by a function of *a* and *b* where *a* is the number of arrows and *b* is the number of boxes in the diagram. The normalisation is implemented as seen in Table 2.

Function	is divided by
gamma_arrows_down	a
gamma_crossing_lines	a^2
gamma_straight_lines	a
gamma_arrow_length	a
gamma_box_line	$a \cdot b$
gamma_overlap	b
gamma_window_size	1
gamma_distance	b^2

Table 2

4.3 Analysis

To test the Simulated Annealing algorithm for this problem, two diagrams have been used: one has only 4 boxes and the other 10 boxes but its structure is less complex. Both have been tested with 300 iterations. To analyse the algorithm's behaviour, both the resulting diagram and the graph showing the development of the cost function will be looked at.

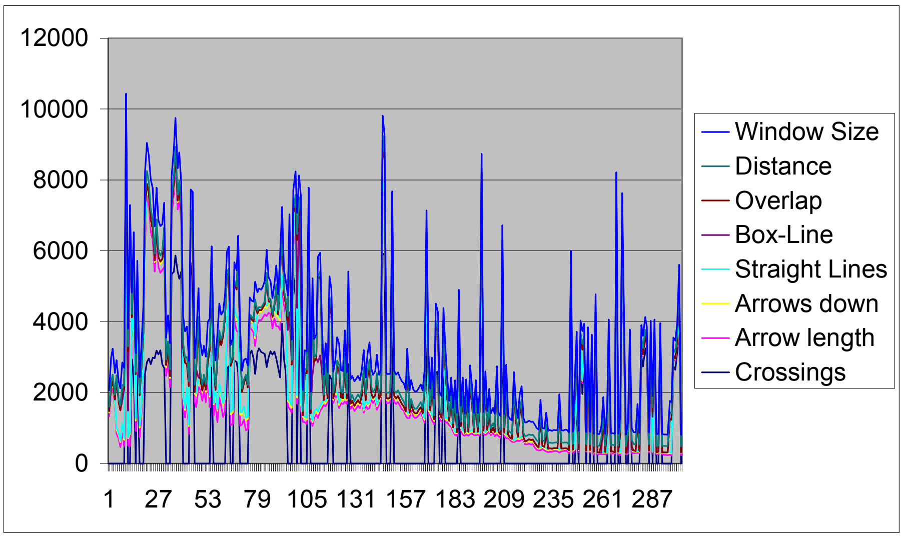

Figure 8 Behaviour of the cost function

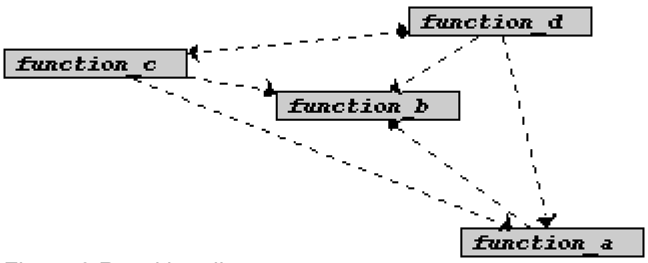

Figure 9 Resulting diagram

Figure 8 shows the cost function as a function of the number of iterations for the diagram displayed in Figure 9. The Γ sub-functions are also shown. Apart from the spikes, which appear to result from the $\Gamma_{crossings}$ function, the cost function converge to a minimal cost. The corresponding diagram is clear and gives an instant overview of the network.

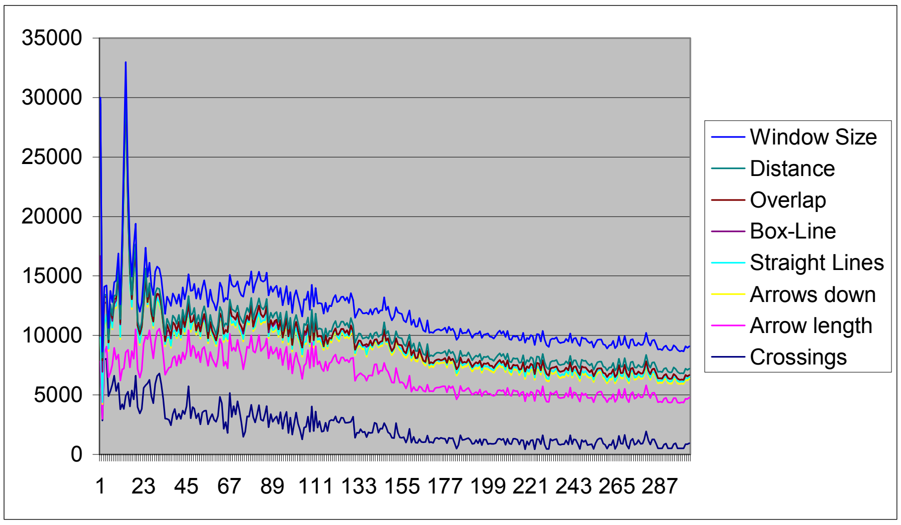

Figure 10 Behaviour of the cost function

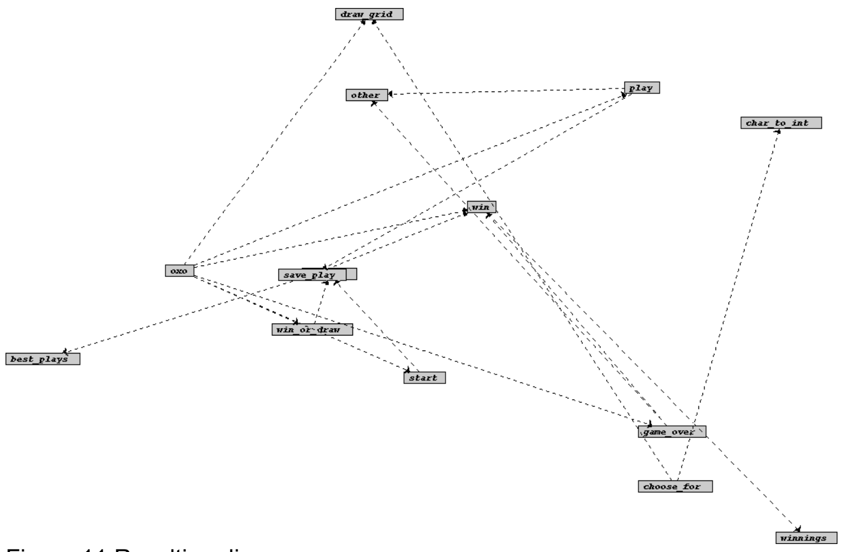

Figure 11 Resulting diagram

Figure 10 shows the cost function for the diagram in Figure 11. Though the cost function seems to converge to a minimum, the diagram is not clear at all. The number of crossings is not minimal and it does not show a clear structure. Furthermore, this diagram took over two hours to be sorted out.

The reason the second diagram was not arranged in a useful way is that too few iterations where performed. A diagram with more boxes needs many more iterations to end up satisfactorily lain out. This is because every new box added to a diagram adds a increasingly large number of new possibilities for arranging a suitable layout. As the Simulated Annealing algorithm works randomly, it is able to reduce that figure. This, however, still implies that every box added to the diagram multiplies the number of iterations needed to obtain a good diagram with a factor, which means the complexity the Simulated Annealing algorithm is exponential.

5 The Sugiyama Method

In the following sections another approach to the problem of automated layout of graphs will be presented. This approach is based on regarding Clarity network diagrams as multilevel directed graphs, or *hierarchies*. This is justified when it is considered that the functions in a Clarity network diagram do not form random structures, but contain some implicit logic. The logic assumed in this approach is that the functions form a hierarchy.

5.1 Theory

This section will illustrate the hierarchical approach specified by Sugiyama [2].
An overview will be given of the appropriate definitions, and the algorithm will be described.
The method is based on the following four steps:

Step I A "proper" hierarchy is formed from a given set of directed pairwise relations among elements of a system. If the hierarchy has long span edges, t is converted into a proper hierarchy by adding dummy vertices and edges.

Step II The number of crossings of edges in the proper hierarchy is reduced by permuting orders of vertices in each level.

Step III Horizontal positions of vertices are determined by considering readability constraints 4 and 5 (see section 3.1). The order of the vertices determined in Step II is given as constraints to preserve the reduced number of crossings.

Step IV A (two-dimensional) picture of the hierarchy is automatically drawn where the dummy vertices and edges are deleted and the corresponding long span edges are regenerated.

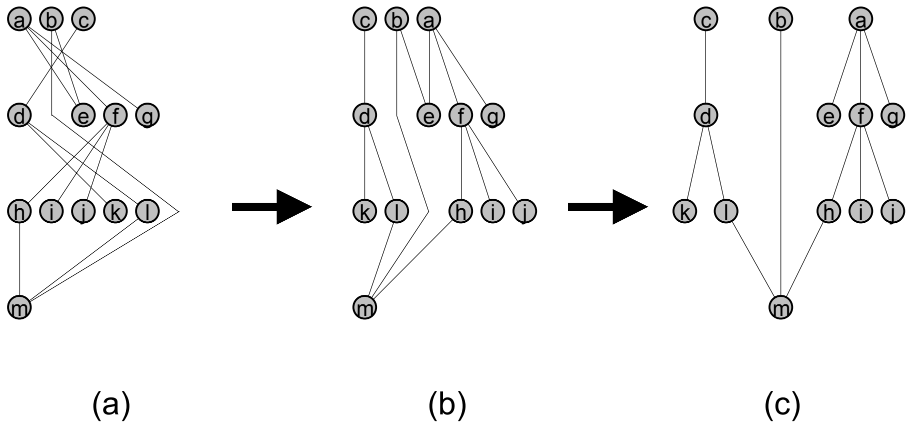

Figure 12 An example of improvements of a drawing

5.1.1 Definitions

n-Level Hierarchy and Map

An *n*-level hierarchy *($n \geq 2$)* is defined as a directed graph *(V, E)* where *V* is the set of vertices and *E* is the set of edges, which satisfies the following conditions.

1. *V* is partitioned into *n* subsets

$$V = V_1 \cup V_2 \cup \cdots \cup V_n \quad (V_i \cap V_j = \varnothing v, i \neq j)$$

where V_i is called the *i* th level, and *n* the length of the hierarchy.

2. Every edge $e = (v_i, v_j) \in E$ where $v_i \in V_i$ and $v_j \in V_j$, satisfies $i < j$, and each edge in E is unique.
   The n-level hierarchy is denoted by $G = (V, E, n)$.
   An n-level hierarchy is called *proper* when it also satisfies the following conditions.
3. E is partitioned into $n - 1$ subsets, that is

   $$E = E_1 \cup E_2 \cup \cdots \cup E_{n-1} \quad (E_i \cap E_j = \emptyset, i \neq j)$$

   where $E_i \subset V_i \times V_{i+1}, i = 1, \cdots, n-1$
4. An order σ_i of V_i is given for each i, where the term *order* means a sequence of all vertices of V_i; $\sigma_i = v_i v_2 \cdots v_{|V_i|}$. The n-level hierarchy is denoted by $G = (V, E, n, \sigma)$, where $\sigma = (\sigma_1, \cdots, \sigma_n)$.

The drawing of a hierarchy is called a map. In a map all the vertices belonging to the i th level V_i are arranged on the i th line of the n horizontal real lines which are numbered from the top to the bottom. The coordinate $x(v_i)$ of vertex v_i on a real line is called a horizontal position. Every edge is drawn with a straight line. By specifying the rules of drawing, the problem is significantly simplified since the number of crossings of a proper hierarchy is determined by order σ of vertices in each level and vertical coordinates of vertices are fixed according to the level where each vertex is included. In subsequent sections discussions are restricted to proper hierarchies. Accordingly, a proper hierarchy is called a hierarchy for simplicity.

Matrix Realization of n-Level Hierarchies

For an n-level hierarchy $G = (V, E, n, \sigma)$, the matrix realisation of G is defined as follows.

1. A matrix $M^{(i)} = M(\sigma_i, \sigma_{i+1})$ is a $|V_i| \times |V_{i+1}|$ matrix whose rows and columns are ordered according to σ_i and σ_{i+1}, respectively.
2. Let $\sigma_i = v_i \cdots v_k \cdots v_{|v_i|}$ and $\sigma_{i+1} = w_1 \cdots w_l \cdots w_{|V_{i+1}|}$. Then the (v_k, w_l) element of $M^{(i)}$, denoted by $m_{kl}^{(i)}$, is given by

   $$m_{kl}^{(i)} = \begin{cases} 1 & \text{if } (v_k, w_l) \in E_i \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

   where $M^{(i)}$ is called an interconnection matrix.
3. A matrix realisation g of G is given by the formula

   $$g(\sigma_1, \cdots, \sigma_n) = M^{(1)} \cdots M^{(n-1)} \quad (= g(V, E, n, \sigma)) \tag{2}$$

The Number of Crossings of n-Level Hierarchies

In the i th interconnection matrix $M^{(i)} = m(\sigma_i, \sigma_{i+1})$ of g, let $\sigma_i = v_1 \cdots v_j \cdots v_k \cdots v_{|V_i|}$. Further, let the row vector of $M^{(i)}$ corresponding to a vertex $v \in V_i$ be denoted by $r(v)$, then the number of crossings $k(r(v_j), r(v_k))$ produced by the ordered pair of row vectors $(r(v_j), r(v_k))$ is given by the formula

$$k(r(v_j), r(v_k)) = \sum_{\alpha=1}^{q-1} \sum_{\beta=\alpha+1}^{q} m_{j\beta}^{(i)} m_{k\alpha}^{(i)} \tag{3}$$

where $q = |V_{i+1}|$. Consequently the formula

$$K(M^{(i)}) = \sum_{j=1}^{p-1} \sum_{k=j+1}^{p} k(r(v_j), r(v_k)) \\ = \sum_{j=1}^{p-1} \sum_{k=j=1}^{p} \left(\sum_{\alpha=1}^{q-1} \sum_{\beta=\alpha+1}^{q} m_{j\beta}^{(i)} m_{k\alpha}^{(i)} \right) \tag{4}$$

gives the number of crossings of $M^{(i)}$ where $p = |V_i|$. Similar expression can be obtained starting from ordered pairs of column vectors. From (4), the total *K(g)* of crossings of *g* is given by

$$K(g) = K(M^{(i)}) + \cdots + K(M^{(n-1)}) \tag{5}$$

Connectivity

In an *n*-level hierarchy $G = (V, E, n, \sigma)$ if $\sigma_i = v_1^i \cdots v_k^i \cdots v_{|V_i|}^i$, $i = 1, \cdots, n$, then the upper connectivity C_{ik}^U of vertex v_k^i and the lower connectivity C_{ik}^L of vertex v_k^i are defined by the formulas

$$C_{ik}^U = \sum_{j=1}^{|V_{i-1}|} m_{jk}^{(i-1)}, \qquad k = 1, \cdots, |V_i|, \quad i = 2, \cdots, n \tag{6}$$

$$C_{ik}^L = \sum_{l=1}^{|V_{i+1}|} m_{kl}^{(i)}, \qquad k = 1, \cdots, |V_i|, \quad i = 1, \cdots, n-1. \tag{7}$$

Barycentres

Barycentre D_y of a binary vector $y = (y_1, \cdots, y_m)$ is given by

$$D_y = \sum_{j=1}^{m} j \cdot y_j \Big/ \sum_{j=1}^{m} y_j . \tag{8}$$

The formulas

$$B_{ik}^R = \sum_{l=1}^{q} l \cdot m_{kl}^{(i)} \Big/ \sum_{l=1}^{q} m_{kl}^{(i)}, \qquad k = 1, \cdots, p \ (= |V_i|) \tag{9}$$

$$B_{il}^R = \sum_{k=1}^{p} k \cdot m_{kl}^{(i)} \Big/ \sum_{l=1}^{p} m_{kl}^{(i)}, \qquad 1 = 1, \cdots, q \ (= |V_{i+1}|) \tag{10}$$

give row and column barycentres of a binary interconnections matrix $M^{(i)} = (m_{ml}^{(i)})$ respectively, which will be used in BC method. Upper and lower barycentres B_{ik}^U, B_{ik}^L of upper and lower vertices connected to the *k*th vertex v_k^i in the *i*the level are defined by

$$B_{ik}^U = \sum_{j=1}^{p} x\left(v_j^{i-1}\right) m_{jk}^{(i-1)} \Big/ C_{ik}^U, \qquad k = 1, \cdots, |V_i| \tag{11}$$

$$B_{ik}^L = \sum_{j=1}^{q} x\left(v_l^{i+1}\right) m_{kl}^{(i)} \Big/ C_{ik}^U, \qquad k = 1, \cdots, |V_i| \tag{12}$$

where $p = |V_{i-1}|$ and $q = |V_{i+1}|$, and $x(v)$ is the horizontal position of a vertex *v*. These barycentres will be used in the PR method.

5.1.2 Creating a proper hierarchy

Determining the level of the vertices

From a directed graph without cycles, an *n*-level hierarchy may be constructed. First, the minimum level of every vertex must be determined:

$$m(v_i) = \begin{cases} \max_{\forall (v_j, v_i) \in E} m(v_j) + 1 & \exists j(v_i, v_j) \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

Some vertices may be shifted down, however, to make the distance between nodes smaller. The level of a vertex is defined as:

$$l(v_i) = \begin{cases} \min_{\forall (v_i, v_j) \in E} l(v_j) - 1 & \exists j(v_i, v_j) \\ m(v_i) & \text{otherwise} \end{cases} \tag{14}$$

The *i*th level of the *n*-level hierarchy is then defined as

$$V_i = \{v \mid l(v) = i\} \tag{15}$$

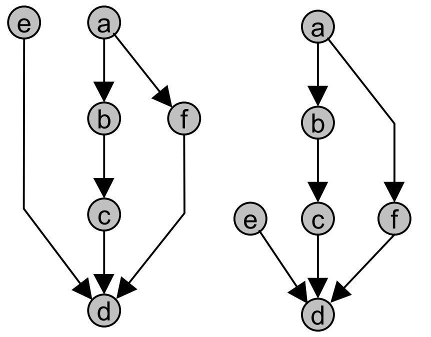

Figure 14 Some vertices may be shifted down

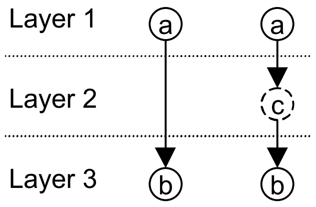

Figure 13 Adding a dummy node

Making the hierarchy proper

Let the directed graph (*V*, *E*) be an *n*-level hierarchy. As a proper hierarchy has only edges $e = (v_i, v_j)$ where $j = i + 1$, *dummy nodes* have to be created for edges that span multiple levels.

Every edge $e = (v_i, v_j)$ where $j > i + 1$ is deleted and replaced by two edges (v_i, v_k) and (v_k, v_j) where $i < k < j$, and vertex v_k is added to V_k. This is repeated until (*V*, *E*) is a proper hierarchy.

($v_i \in V_i$ throughout)

5.1.3 Reducing the number of crossings using the Barycentric Method

This section will explain how *Step II* mentioned in section 5.1 is achieved.

The problem as formulated by Sugiyama [2] is as follows. Let S_i be a set of all possible orders σ_1 in an *n*-level hierarchy $G = (V, E, n, \sigma)$ and $S = S_1 \times \cdots \times S_n$, then the problem of minimizing the number of crossings of the *n*-level hierarchy is stated as follows:

$$\text{minimize}\{K(g(\sigma)) \mid \sigma \in S\} \tag{16}$$

This problem is combinatorial in nature, which makes it difficult to obtain the optimum solution when the size of the problem is not small.

The *Barycentric method*(BC method) is a heuristic method for the reordering of the row order $\sigma_1 = v_1 v_2 \cdots v_{|V_1|}$ to reduce the number of crossings under the fixed column order σ_2 in $M(\sigma_1, \sigma_2)$. The essence of this method is to reorder σ_1 according to the row barycentres

$B^R_{1k},\ k = 1,\cdots,|V_1|$ which are calculated by (9). These are ordered from the smallest to the largest, that is $B^R_{1s_1} \leq B^R_{1s_2} \leq \cdots \leq B^R_{1s_{|V_1|}}$, where if there are sets of rows of which barycentres are equal, the original order is preserved. Then if the reordering of σ_1 is denoted by σ'_1, we have $\sigma'_1 = v_{s1}v_{s2}\cdots v_{s|V_1|}$. The operation which transforms $M(\sigma_1,\sigma_2)$ to the reordered matrix $M(\sigma'_1,\sigma_2)$, is called *barycentric ordering* of rows and is denoted by β_R, i.e., $M(\sigma'_1,\sigma_2) = \beta_R(M(\sigma_1,\sigma_2))$. The barycentric ordering of columns is similarly defined and is denoted by β_C. The number of crossings can be reduced by repeating the barycentric ordering of rows and columns in turn. A study on the effectiveness of the barycentric ordering method can be found in [2].
The order of rows which have equal barycentres is initially preserved, however it has been found [2] that reversing the order or rows with equal barycentres may reduces the number of crossings. This operation is called *reversion* and is denoted by $R_R(M)$. (or $R_C(M)$ for columns)

The Algorithm

We consider an n-level hierarchy $g(\sigma_1,\cdots,\sigma_n) = M(\sigma_1,\sigma_2)M(\sigma_2,\sigma_3)\cdots M(\sigma_{n-1},\sigma_n)$ where $n \geq 3$.
The algorithm consists of two phases, phase 1 and phase 2. Phase 2 uses phase 1 as a subalgorithm. Phase 2 is executed after phase 1.

Phase 1

```
Step 1: An initial order $\sigma'_1 \in S_1$ is given. Let $i$ := 1.
Step 2: $M(\sigma_i,\sigma'_{i+1}) := \beta_C(M(\sigma_i,\sigma_{i+1}))$
Step 3: If $i < n - 1$, then $i := i + 1$ and goto Step 2. If $i = n - 1$, then goto Step 4.
Step 4: $M(\sigma'_i,\sigma_{i+1}) := \beta_R(M(\sigma_i,\sigma_{i+1}))$
Step 5: If $i > 1$, then $i := i - 1$, and goto Step 4. If $i = 1$, then STOP.
```

Step 2 is called the DOWN procedure, step 4 the UP procedure. The whole procedure i.e. DOWN and UP is iterated an initially given number of times.

Phase 2

```
Step 1: Let $i := n - 1$.
Step 2: $M(\sigma'_i,\sigma_{i+1}) := R_R(M(\sigma_i,\sigma_{i+1}))$
Step 3: Phase 1
Step 4: If $i > 1$, then $i := i - 1$, and goto Step 2. If $i = 1$, then goto Step 4.
Step 5: $M(\sigma_i,\sigma'_{i+1}) := R_C(M(\sigma_i,\sigma_{i+1}))$
Step 6: Phase 1
Step 7: If $i < n - 1$, then $i := i + 1$ and goto Step 4. If $i = n - 1$, then STOP.
```

Step 2 is called the UP procedure, step 4 the DOWN procedure. The whole procedure is iterated an initially given number of times.

5.1.4 Determining horizontal position of vertices using the Priority Method

This section will explain how *Step III* in section 5.1 is achieved.
The Priority Layout Method is a heuristic method which is developed by Sugiyama [2] to reduce the computing cost needed to obtain horizontal positions of vertices that realise a readable layout of a given n-level hierarchy. The fundamental idea for this method is similar to that for the multilevel BC method, i.e., a decomposition of the problem and "sequential" application of level operations (called improvements of horizontal positions). In case of the

multilevel BC method, the reordering of vertices is performed according to barycentres of the vertices, while in the PR method the improvement is carried out according to "priority numbers" given to boxes.

As the PR method uses the absolute positions of the boxes to straighten the arrows and arrows start and end at the centres of the boxes, the centres will be used.

We define $x^i = (x_1^i, \cdots, x_{|V_i|}^i)$ as the positions of the boxes in level *i*, $w^i = (w_1^i, \cdots, w_{|V_i|}^i)$ as the width of the boxes in level *i* (which is given) and *d* as the minimum distance between two boxes.

1. Initial values of horizontal positions of boxes in each level are given by

$$x_k^i = x_{k-1}^i + d + \frac{w_{k-1}^i + w_k^i}{2}, \qquad k = 1, \cdots, |V_i|, \quad i = 1, \cdots, n \tag{17}$$

   where $w_0^i = 0$.

2. Positions of boxes in each level are improved in the order of levels $2, \cdots, n; n-1, \cdots, 1; t, \cdots, n$ where *t* is a given integer ($2 \le t \le n-1$). The improvements of the positions of vertices in levels $2, \cdots, n$ and $t, \cdots, n$ are called DOWN procedures, while those for level $n-1, \cdots, 1$ are called UP procedures.
3. Positions of boxes in each level are determined one by one according to their priority numbers. The highest priority number, an integer more than the maximum of connectivities of all boxes, is given to dummy nodes to improve the readability constraint 5 (section 3.1). Priority numbers given to the boxes are the connectivities of the vertices calculated by (6) or (7) in section 5.1.1.
4. The principles to improve the position of a box is to minimise the difference between the present position of the box and the upper (or lower) barycentre given by (11) (or (12)) in paragraph 0, of the box in DOWN (or UP) procedure under the following conditions.
   a) The box should have a minimum distance of *d* to the other boxes in the same level.
   b) The order of boxes in each level should be preserved (readability constraint 1).
   c) Positions of only boxes of which priorities are less than the priority of the box can be changed, where the distance displaced should be as small as possible (readability constraint 4).

5.2 Implementation

This section will focus on how the Sugiyama method for automated layout design, as described in the sections above, has been implemented in the functional system Faith/Clarity.

5.2.1 Data types

In the implementation of the matrix realisation of a *n*-level hierarchy, as discussed in section 5.1.1, in the Clarity system, a data type called `matrix` is used. Figure 15 depicts the Clarity *constructor* which belongs to this data type. The constructors first argument is the actual binary matrix in the form of a list of list of integers, the other two arguments are lists of integers to keep track of what picture elements (boxes) correspond to the rows (columns) of the matrix. For the implementation of *i* matrices a function `sugiyama_M` is used. This function returns the *i* th matrix when input the integer parameter *i*.

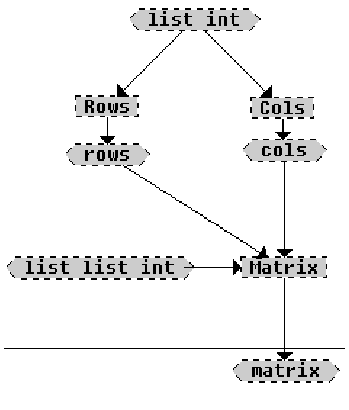

Figure 15 sugiyama_M

5.2.2 The BC algorithm

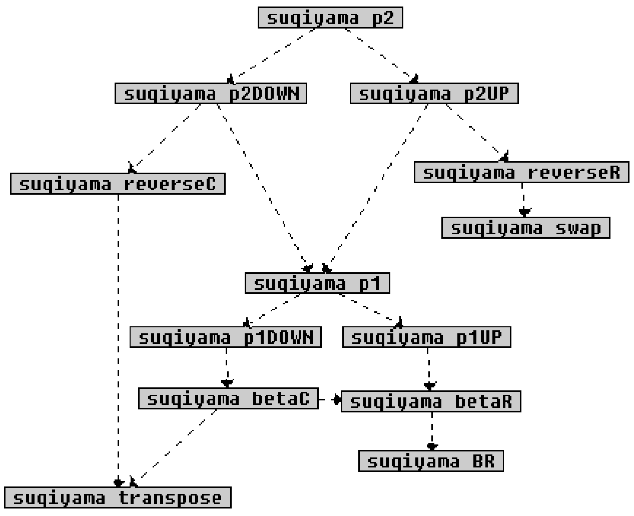

Figure 16 Network diagram of Sugiyama functions

The functions depicted in Figure 16 perform operations on the matrices in order to implement the BC algorithm discussed in section 5.1.3.
The function sugiyama_p1 is the implementation of the BC method Phase 1. It uses sugiyama_p1DOWN and sugiyama_p1UP to implement the corresponding BC method DOWN and UP procedures. sugiyama_p1UP calls sugiyama_betaR for all matrices, starting at $n - 1$, and ending at 1. sugiyama_betaR calculates the row barycentres using sugiyama_BR and orders the rows accordingly (see section 5.1.3) When a barycentre is undefined for a row (i.e. row of all zeros), this row is not reordered. sugiyama_p1DOWN calls sugiyama_betaC for all matrices, starting at 1 and ending at $n - 1$. sugiyama_betaC calculates column barycentres and orders the columns accordingly, by transposing the matrix using sugiyama_transpose and calling sugiyama_betaR. When a barycentre is undefined for a column, this column is not reordered either. Phase 1 (DOWN-UP) is iterated 5 times.
BC method Phase 2 is implemented by the function sugiyama_p2, which uses the functions sugiyama_p2UP and sugiyama_p2DOWN. sugiyama_p2UP, the implementation of the BC method Phase 2 UP procedure, calls sugiyama_reverseR to swap rows with equal barycentres (using sugiyama_swap) and calls sugiyama_p1 to initiate a Phase 1, for all matrices. (starting at $n - 1$ and ending at 1) sugiyama_p2DOWN performs a similar operation for matrix columns. Phase 2 (UP-DOWN) is iterated 3 times.

5.2.3 PR Algorithm

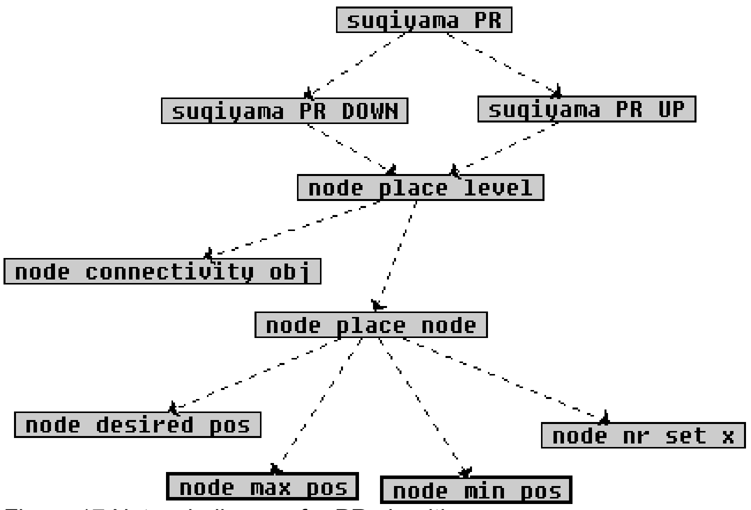

Figure 17 Network diagram for PR algorithm

The network diagram in Figure 17 illustrates the implementation of the PR algorithm in Clarity. sugiyama_PR uses sugiyama_PR_DOWN and sugiyama_PR_UP, which iteratively call node_place_level to reorder a level of boxes. The node_connectivity_obj function calculates the connectivity for a node, which is used to determine its priority.

`node_place_node` is then called, which tries to put the node at its desired position (the barycentre) within the bounds given by `node_max_pos` and `node_min_pos`, which calculate the minimum possible and maximum possible position given the positions of nodes with higher priorities.

5.3 Analysis

In this section, the behaviour of the Sugiyama algorithm is discussed in terms of the reduction of crossings and the adherence to the readability constraints (section 3.1).

5.3.1 Quantitative Analysis

To analyse the behaviour of the Sugiyama algorithm, a test of 22 real-world example Clarity network diagrams has been performed. The number of crossings K_0 in the diagrams after constructing a hierarchy is compared to the number of crossings after executing the BC algorithm for reducing crossings K_{BC}. See (18). The results are shown in Figure 18 and Table 3.

$$\eta = \frac{K_{BC}}{K_0} \qquad (18)$$

Figure 18 is a histogram of the data in Table 3, which shows η between 0.0 and 0.2 was most common, $\eta = 1.0$ ($K_0 = K_{BC}$) occurred only once, and $\eta > 1.0$ ($K_0 < K_{BC}$) also occurred only once. From the mean $\overline{\eta} = 0.41$, the variance $Var(\eta) = 0.23$ and the median $med(\eta) = 0.22$, it can be concluded that in almost all cases the number of crossings will be reduced substantially.

K_0	K_{BC}	η
131	41	0.31
33	26	0.79
2	0	0.00
8	0	0.00
250	84	0.33
66	49	0.74
16	10	0.63
21	17	0.81
5	0	0.00
2	4	2.00
0	0	-
49	9	0.18
16	1	0.06
2	0	0.00
3	0	0.00
4	0	0.00
0	0	
13	2	0.15
158	34	0.22
0	0	-
56	56	1.00
47	22	0.47

Table 3 Calculated values for η

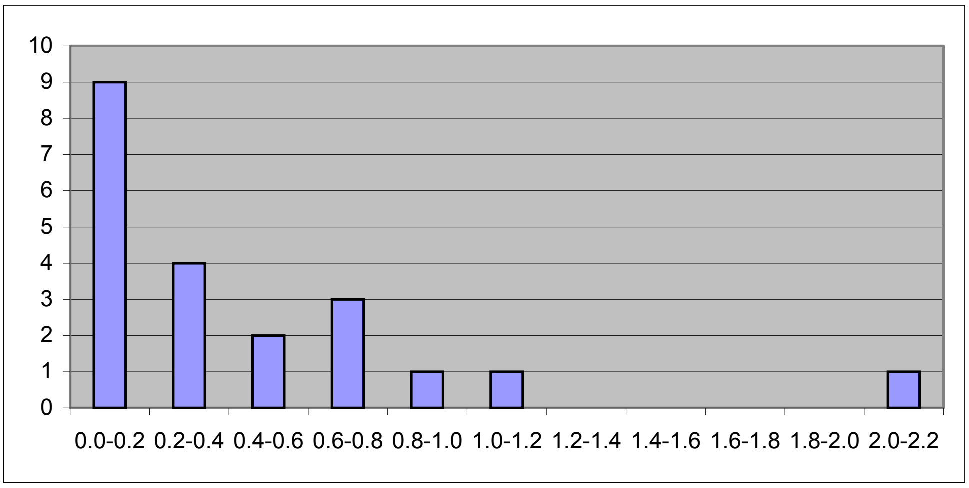

Figure 18 Histogram of the data in Table 3

5.3.2 Qualitative Analysis

To test the structural and aesthetical aspects of the Clarity network diagrams produced by the Sugiyama algorithm, the algorithm has been applied to several diagrams. For brevity, only one example is shown, though the remarks are in this section generally applicable.

Figure 19 shows the original diagram as made by a user. Figure 20 shows the same diagram after the Sugiyama algorithm has been applied to it. The readability constraints of network diagrams, as formalised in section 3.1, will be considered.

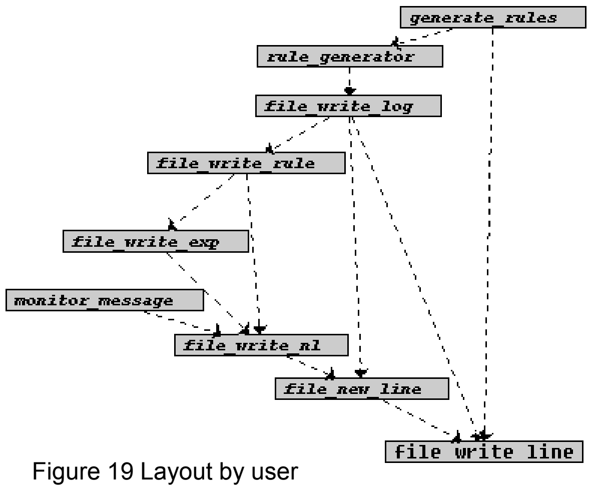

Figure 19 Layout by user

1. The number of crossings should be minimised. This is performed by the BC method.
2. Lines should not cross boxes. The Sugiyama algorithm places boxes on levels and the PR method distributes them on the level. Dummy nodes are inserted when lines join boxes with one or more layers between them.
3. As the algorithm is based on regarding Clarity network diagrams as hierarchical structures, hierarchies manifest themselves as levels in the diagram produced.
4. Arrows should be short. The hierarchical approach ensures minimal arrow length between boxes on adjacent levels. This, however, does not hold for arrows between boxes that are not on adjacent levels.
5. It is desirable that arrows are either vertical or horizontal. This is dealt with by the Sugiyama PR algorithm for arrows that span multiple levels. Short arrows are not optimised for straightness, however.
6. It is desirable that the distance between any two boxes is short. Though connected boxes are kept together, unconnected boxes may be placed at a distance.
7. It is desirable that the area a network takes up is small. Though the distance between different levels and the distance between boxes in levels can be changed, the strict layout in levels inherently makes for more levels than absolutely necessary, and the levels inherently take up more space than absolutely necessary. However, within the constraints imposed by the levels, the total space needed is minimised.

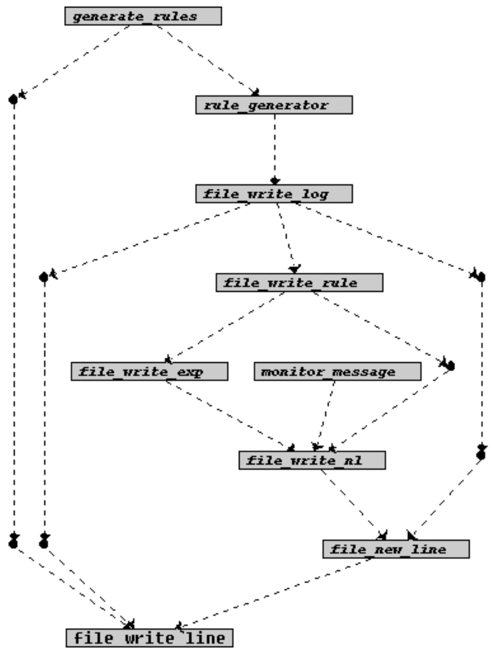

Figure 20 Layout by the algorithm

The Sugiyama algorithm handles readability constraints 1 to 3 well, though constraints 4 to 7 are only partially dealt with. The constraints 4 to 7, however, have lower priority than the other constraints (see Table 1).

6 Analysis

Two different approaches to the problem have been proposed. In this section, they will be compared to see which creates the most satisfactory diagrams. An example diagram will be used, which is the same diagram that has been used in section 4.3.
On the test system, the Simulated Annealing algorithm needed more than two hours to rearrange the diagram, while the Sugiyama algorithm took only 25 seconds.
Simulated Annealing is used to minimise a function, which is in this case the cost function Γ. The result of the cost function for the diagram found by the algorithm was 8615, while the cost function returned 1334 for the layout made by the Sugiyama algorithm.
Furthermore, the latter network diagram gives an immediate overview, while the former is a mess.

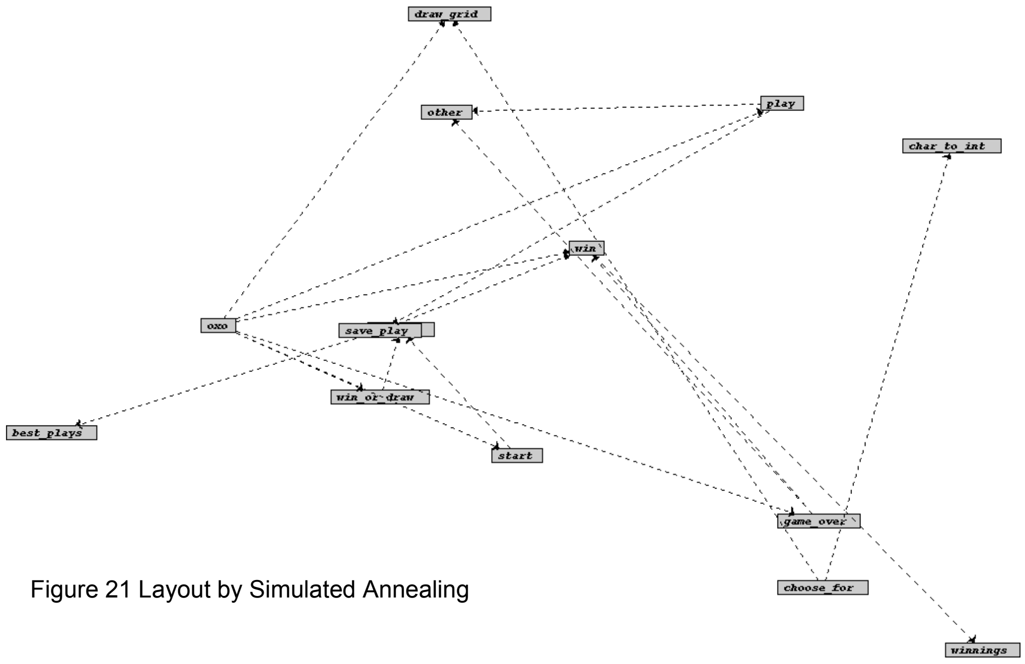

Figure 21 Layout by Simulated Annealing

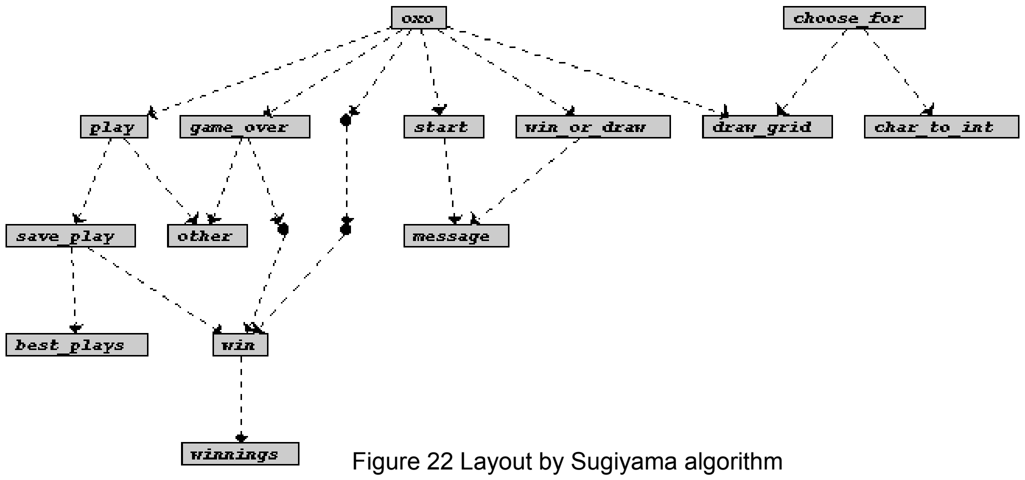

Figure 22 Layout by Sugiyama algorithm

Concluding, the Sugiyama algorithm produces the most satisfactory diagrams.

6.1 Simulated Annealing

Advantages and disadvantages of Simulated Annealing have been discussed by Tamassio [3], who sums up the following advantages:

- Relatively simple to implement. No difficult algorithms are needed apart from the Simulated Annealing algorithm itself. All other requirements are implemented using the cost function.
- Smooth evolution of drawing into the final configuration helps to preserve the user's mental map of the diagram.
- Works well in practice for small graphs. Small graphs can be sorted out in relatively short time, whereas larger diagrams take more time to rearrange..

He sums up the following disadvantages:

- Slow running time. Because of the random nature of the method, calculating a satisfactory layout may take a long time.
- Few theoretical results on the quality of the drawings produced. The Sugiyama method, for example, reduces the number of crossings, while the Simulated Annealing algorithm minimises an arbitrarily defined function.
- Limited constraint satisfaction capability. Constraints may be implemented using the cost function, but this does not guarantee that constraints are satisfied.

6.2 Sugiyama method

Advantages of algorithmic approaches have been discussed in section 3.2. Another advantage to using the Sugiyama method to rearrange Clarity network diagrams is that computer programs contain an implicit logic that the Sugiyama method assumes to be the presence of hierarchical structures.

Because recursive functions are not often used in computer programs, because they use much memory, cycles are not commonly found in network diagrams. Therefore, in general functions can be assigned a level in a hierarchy and Clarity network diagrams do form hierarchies. The Sugiyama method uses this to arrange the network diagram in a satisfactory layout.

7 Conclusions and Recommendations

Research has been done on finding a method for automated graph layout that can be used in conjunction with the visual development environment Clarity, so that users do not have to arrange Clarity network diagrams themselves. There are two reasons why the Sugiyama method is preferred above the other method discussed, Simulated Annealing:

- The running time the Sugiyama algorithm takes to produce a satisfactory layout is much less than the time needed for the Simulated Annealing algorithm to achieve the same; furthermore, the running time needed for larger diagrams increases exponentially for the Simulated Annealing algorithm, and polynomially for the Sugiyama algorithm.
- Layouts produced by the Sugiyama algorithm give an immediate insight in the hierarchical structure of a diagram by producing a clear separation in levels.

Because of the non-random nature of the Clarity network diagram, Simulated Annealing is not a inadequate method as such, but the Sugiyama method addresses the specific needs for Clarity network diagrams. Therefore, it is faster and produces better results.

8 References

[1] Christensen et al, 1995: “Algorithms for Cartographic Label Placement”, in *Proceedings of the American Congress on Surveying and Mapping I*, pages 75-89, 1993.

[2] K. Sugiyama, S. Tagawa, and M. Toda, “Methods for Visual Understandings of Hierarchical System Structures”, in *IEEE Transactions in Systems, Man, and Cybernetics*, vol. smc-11, no.2, pp. 109-125, February 1981.

[3] R. Tamassia, I. F. Cruz, “Graph Drawing Tutorial”, *http://www.cs.brown.edu/people/rt/papers/gd-tutorial/gd-constraints.pdf*.